Scheduling Multi-Component Applications Across Federated Edge Clusters With Phare

Received 10 January 2024; revised 25 February 2024; accepted 12 March 2024. Date of publication 18 March 2024; date of current version 2 April 2024.

Digital Object Identifier 10.1109/OJCOMS.2024.3377917

Scheduling Multi-Component Applications Across Federated Edge Clusters With Phare

GABRIELE CASTELLANO[1], STEFANO GALANTINO[2], FULVIO RISSO[2], AND ANTONIO MANZALINI[3]

[1]Centre Inria d'Université Côte d'Azur, 06902 Valbonne, France
[2]Department of Computer and Control Engineering, Politecnico di Torino, 10129 Turin, Italy

[3]Innovation Labs, Telecom Italia Mobile, 10148 Turin, Italy

CORRESPONDING AUTHOR: S. GALANTINO (e-mail: stefano.galantino@polito.it)

This work was supported in part by the European Union's Horizon Europe Research and Innovation Programme Project FLUIDOS (Flexible, Scalable, Secure, and Decentralised Operating System) under Grant 101070473. The work of Stefano Galantino was supported by TIM S.p.A. through the Ph.D. Scholarship.

ABSTRACT The shift towards agile microservice architecture has enabled significant benefits for IT companies but has also resulted in increased complexity for Cloud orchestration tools. Traditional tools were designed for centralized data centers and are ineffective for locating microservices in geographically-distributed edge-like infrastructures. This paper presents Phare, a decentralized scheduling algorithm designed to optimize the placement of microservices by satisfying their computing and communication demands while minimizing deployment costs. Phare employs a heuristic-based approach to solve the NP-Hard scheduling problem, prioritizing the microservices with the more stringent requirements and placing them on the most convenient computing facilities, based on the concept of *affinity*, contributing to the field by providing a more holistic approach to resource scheduling in edge computing. We validate our approach against Firmament, the state-of-the-art workload scheduling algorithm for component-based applications, on simulated edge infrastructures with hundreds of clusters. Phare achieves up to a 10× reduction in terms of deployment costs compared to Firmament while providing a much lower scheduling latency.

INDEX TERMS Resource sharing, cloud-to-edge, service allocation.

I. INTRODUCTION

IN THE last decade, we experienced a paradigm shift in Web application development patterns, moving from huge monolithic frameworks to the agile microservice approach. The strict decoupling of application logic into small, dedicated components enabled substantial benefits for IT companies both in terms of quality of experience provided to the end-user (*QoE*), and cost savings for *DevOps* practices. Although providing enhanced scalability and resiliency upon unexpected disruptive events, such application decoupling also resulted in increased complexity for traditional Cloud orchestration tools like Kubernetes [1] and Hadoop [2], [3]. Specifically, data center scheduling algorithms must ensure at any time to match microservice specifications in terms of SLOs, reserving computing and networking resources for their execution. Customers will then be charged based on the requested amount of resources (CPU, memory, disk, bandwidth, etc.) and the related guarantees in terms of service availability.

The complexity of the scheduling process lies *(i)* in the heterogeneity of microservice resource requirements, and *(ii)* in the additional limitation derived from modern data center architectures. In fact, until recently, major Cloud providers scaled up their computing facilities by building "mega-DCs" with hundreds of thousands of servers and interconnecting them into a wide-area backbone. However, a different scaling strategy has quickly become standard, shifting from "mega-DCs" to using a collection of smaller DCs located within close proximity. This shift is driven by two pressures: *(i)* the difficulty of siting and provisioning large facilities; and *(ii)* the desire for fault tolerance to survive an outage in a single location [4], [5].

Additionally, recent trends towards Edge, Fog, and Liquid [6] computing solutions favored, even more, the geo-distribution of computing facilities [7], [8] in the attempt to guarantee the most appropriate hosting infrastructure for latency-sensitive applications. With the advent of Edge

computing, telecommunication networks, IoT systems, and Smart City/Grid technologies have significantly enhanced their operational efficiency and resilience. These advancements may leverage thousands of distributed computing resources, markedly improving the Quality of Experience (QoE) and system robustness [9].

We argue that such Cloud solutions are far from behaving effectively when trivially adapted to the Edge scenario. Indeed, these solutions have been designed for centralized data centers, with guarantees of computing and network resources, and are not designed to identify suitable microservice placement considering their communication patterns. Therefore, they fail to scale on geographically distributed edge-like infrastructures seamlessly, specifically when dealing with nodes that are geographically spread over high-latency WANs [10], [11], [12].

Furthermore, the extreme dynamicity of microservice resource usage patterns drastically increases the complexity of the scheduling process (e.g., the workload may vary depending on the number of users connecting to the Web application). Nevertheless, it is still possible to roughly differentiate them based on their *expected* execution time: production clusters deploy a huge variety of long-running applications (*LRAs*), long-lived microservices that continue execution for days to months. LRAs are commonly used for stream processing [13], [14], [15], Web services [16] and machine learning tasks [17], [18], [19], and recent work estimated that a substantial portion of production cluster – ranging from 10% up to 50% – is entirely dedicated to LRAs workloads [20], [21]. In comparison, conventional offline batch processing workloads (e.g., Spark and MapReduce jobs) run short-lived tasks that typically finish within minutes or shorter. Long-running applications can typically withstand longer scheduling times, but they require optimal placement, whereas short-running applications are latency-sensitive. Scheduling algorithms must then be able to handle both LRAs and SRAs, trading off between the different requirements. The scheduling process of such heterogeneous workloads, accounting not only their computing requirements (i.e., CPU, RAM, GPU share, and more) but also the networking requirements (i.e., communication bandwidth), while minimizing the deployment cost is — to the best of our knowledge — still unexplored.

In this paper, we argue that adapting Cloud scheduling solutions to the Edge case is not effective and leads to suboptimality in practice. To this end, we present Phare, a decentralized scheduling algorithm that places microservices on geographically distributed infrastructures. Such distributed computing facilities constitute what we refer to as a *federation*. Each constituent part of the federation (i.e., a cluster) offers (a subset of) its computing resources to the other members of the federation, allowing each individual and possibly autonomous entity to purchase resources when needed, creating a continuum of heterogeneous computing resources [6]. The primary objective of Phare is to optimize the execution of microservices by meeting their computing and communication needs while minimizing deployment costs. To accomplish this, we design a heuristic-based algorithm to solve the NP-Hard scheduling problem, and we evaluate the performance of Phare against Firmament [22], the Kubernetes state-of-the-art scheduling algorithm, on simulated federated infrastructures with hundreds of clusters. Our approach achieves almost a 10× reduction in terms of deployment costs compared to Firmament while always guaranteeing a lower scheduling latency.

The rest of the paper is organized as follows. Section II summarizes related work, Section III formalizes the cost minimization problem and Section IV proposes the heuristic-based algorithm Phare. Finally Section V extensively evaluates the performance of Phare on simulated environments and Section VI concludes the paper.

II. RELATED WORK

The problem of scheduling in Cloud computing has been deeply addressed in the last two decades, while only a few, more recent, solutions address the additional challenges that arise in Edge computing. The most adopted solutions for container orchestration, such as Kubernetes [1] and YARN [3], provide generic scheduling algorithms, responsible for placing jobs on the available machines, and have been designed to address a large portion of common use cases while balancing complexity, scheduling latency, and optimality. While such algorithms may effectively solve the scheduling problem in a traditional Cloud environment, characterized by homogeneous resources, adapting them to distributed Edge infrastructures may not be trivial.

In literature, jobs are typically classified based on the *expected* execution time: long jobs (LRA, Long Running Application) tend to be latency-insensitive and require near-optimal placement, as they are expected to run for days or even months, whereas short jobs (SRA, Short Running Application) are latency-sensitive, and typically finish within minutes or less. Consequently, especially in production environments, scheduling algorithms must deal with both SRAs and LRAs, providing a trade-off among the above requirements.

The problem of scheduling SRAs has been widely addressed in the literature, leveraging task reordering techniques to prevent head of line blocking [23], [24], and introducing also task bandwidth requirements to cope with the most network demanding tasks [25], [26], [27]. Still, inaccurate estimates of job completion time can be difficult to mitigate due to external factors such as data size, network congestion, and resource contention which make expected completion time highly variable.

While most relevant and recent works on Edge Computing focus on SRA scheduling, the problem of scheduling LRA, such as micro-service based applications, is still overlooked to the best of our knowledge. For this kind of problem, the focus moves from completion time to deployment optimality in terms of the final deployment cost and the efficient usage of both computing and networking resources. The

problem has been however widely addressed in the context of data centers since public Cloud computing has emerged as the most promising solution to host companies' IT services. A simple and flexible family of algorithms handles the problem one job per time, i.e., each unscheduled job is first retrieved from a queue and then assigned to a computation unit regardless of the other jobs that are still in the queue [23], [28]. This approach has the limitation of committing early to suboptimal decisions that can prevent the placement of subsequent jobs. To overcome such limitations, some solutions jointly process batches of tasks. For instance, Stratus [29] proposes an algorithm that targets the IaaS (Infrastructure as a Service) scenario; specifically, it aims to maximize the use of the purchased resources by co-allocating tasks onto the same VMs. Quincy [30] introduces the concept of *flow scheduling*, where the problem of job scheduling is converted to an equivalent *min-cost max-flow* problem. Such an approach is further improved by Firmament [22], which achieves the same high-quality deployments but at a much faster scheduling time. Firmament is currently adopted in widespread Kubernetes clusters and can efficiently minimize the overall application deployment cost while horizontally scaling up to thousands of servers.

Although very promising, all the solutions above have been designed to address a Cloud-like environment and do not account for the additional challenges of an edge infrastructure. In particular, inter-job communication may feature bandwidth requirements that are not trivial to satisfy: a series of new constraints can make such models ineffective, and, nonetheless, the communication requirements may lead to additional inter-cluster network costs based on the final job placement. As we will show in Section V, it can be hard to cope with such additional problems by simply extending/adapting well-established cloud scheduling algorithms.

Motivated by the heterogeneity of resources and, therefore, of constraints that may affect the job placement at the edge of the network, a set of recent works addresses the problem in terms of inter-job dependencies, proposing the so-called *rule-based scheduling* [1], [3], [31]. Domain experts provide a qualitative representation of the *interferences* between jobs, which can be in terms of reciprocal affinity and anti-affinity. Then, the scheduling decision takes into account such information and places jobs accordingly. However, these approaches only take into account qualitative information, failing to capture and optimize quantitative effects on the cluster performance. Medea [20] tries to overcome such limitations by providing a highly expressive model to describe the job requirements; such an algorithm ensures low latency placements and enables cluster owners to specify enhanced placement constraints for long-running containers. Although the improved expressiveness guarantees better scheduling modeling, it still relies on experts to summarize the sophisticated interference.

As a further optimization, network-aware resource management strategies integrate data center topology information and/or application characteristics. [32], [33], [34], [35] focus

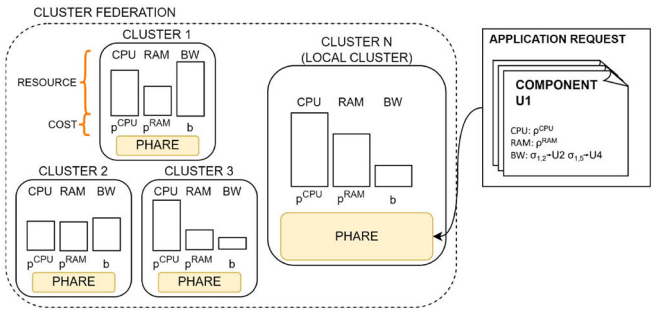


FIGURE 1. High level overview of the Phare architecture.

on Integer (Non) Linear Programming (ILP/INLP) models to find the optimal allocation scheme based on an optimization objective. Although able to identify optimal placements, these solutions cannot find a feasible solution within an acceptable time, thus limiting their applicability in production environments. The computational complexity can be reduced by either decomposing the optimization problem into parallel tractable INLP subproblems [36] or limiting the search space to a subset of compute nodes, based on the concept of open subscriber group mode [37] to balance quality of allocation and convergence time. The high-dimensional search space can be further reduced considering only a subset of the possible requirements, allocating MapReduce tasks [38] or data-parallel distributing deep learning jobs [39] based solely on the network topology, selecting the most promising SmartNIC-Accelerated Server based on the compute demand (i.e., CPU and memory) of microservice-based applications [40], or allocating network intensive tasks on geographically distributed edge-to-cloud infrastructures [41], [42]. Still, few approaches effectively account for computing and networking resources, while minimizing the application deployment cost within a reasonable time.

Finally, in the last couple of years, researchers proposed various approaches to address the various challenges that arise in the Edge computing scenario, such as joint scheduling of computing and networking resources [43], [44], distributed scheduling in multi-provider environments [45], [46], support for mobility [47] and energy efficiency [48]. However, proposed solutions are still in an embryonal stage and far from guaranteeing the same scalability properties as cloud-oriented solutions (e.g., as Firmament does). In our work, we provide an enhanced scheduling model that *(i)* overcomes the limitation of cloud-based approaches by providing both qualitative and quantitative measures of inter-job interactions, and *(ii)* enables a highly scalable algorithm that can quickly schedule complex applications on thousands of nodes.

III. SYSTEM MODEL

We consider a distributed edge infrastructure where resources are grouped into clusters. Potentially, each cluster $v \in \mathcal{N}$ is owned by a different edge provider and participates in what we call cluster federation (see Fig. 1). Clusters are heterogeneous and may provide different resource capabilities

(e.g., centralized data centers, network access base stations, central offices, but also isolated user devices). In this work, we consider capabilities in terms of *computing resources* (e.g., the total amount of CPU and memory available in the cluster), and *communication resources* (i.e., amount of network bandwidth used to communicate with other clusters). Since different types of computing resources experience similarities in terms of provisioning and cost evaluation, we define Γ as the set of all computing resources (e.g., CPU and RAM), and treat every $r \in \Gamma$ jointly, as the subsequent steps need to be evaluated for each one of them. On the other hand, we differentiate the notation for communication resources, as we need to treat them separately in our model. In particular, on cluster $v \in \mathcal{N}$, we denote with $Crv \in \mathbb{R}_+$ the budget of computing resource of type r, and with $Nv \in \mathbb{R}_+$ the budget of communication resources.

Requests for deploying applications are issued to edge providers. Each application $i \in \mathcal{I}$ consists of a set of components $\mathcal{M}_i \subseteq \mathcal{M}$, where $\mathcal{M}$ is the set of all possible components. Each component $j \in \mathcal{M}_i$ features resource demands both in terms of computing $\rho_j^r \in \mathbb{R}_+$ (required amount of computing r-resource) and communication with other components $\sigma_{j,k} \in \mathbb{R}_+$ (bandwidth required by j to communicate with component k from the same application).

Edge providers jointly deploy applications across their clusters, thus forming a federated edge infrastructure. Upon receiving the request for deploying an application, the concerned edge provider decides which of the application components should be executed locally (i.e., on its own cluster) and which of them will instead be offloaded to foreign clusters across the federation.

Each type of resource r that is available on a certain cluster features a given price per unit. To preserve generality, we assume that resources may be exposed with different prices to different partners of the federation. We denote with $p_{v,v'}^r \in \mathbb{R}_+$ and $b_{v,v'} \in \mathbb{R}_+$ respectively the unitary price of computing resource r and communication resources on cluster v as seen by the provider of cluster v'. We denote by

$$x_{j,v}^i \in \{0, 1\}, \text{ for } i \in \mathcal{I}, j \in \mathcal{M}_i, v \in \mathcal{N}, \tag{1}$$

the decision variable that indicates if component j from application i has been scheduled on cluster v for deployment. When deploying a certain component $j \in \mathcal{M}_i$ on cluster v, edge provider v' experiences a cost given by multiplying the amount of each demanded resource for the unitary price seen on the hosting cluster:

$$\mathcal{C}_{v'}(j, v) = \sum_{r \in \Gamma} \rho_j^r p_{v,v'}^r + \sum_{k \in \mathcal{M}_i} \sigma_{j,k} b_{v,v'} \mathbb{1}_{\{x_{k,v}^i \neq 1\}}. \tag{2}$$

Note that the cost $\sigma_{j,k} b_{v,v'}$ due to the communication between components j and k is accounted only if j and k are not deployed on the same cluster.

When scheduling components of application i on available clusters, the Edge Provider seeks cost minimization of the overall deployment, and its decision is subject to the resource constraints of the federated edge infrastructure. We formulate such optimization problem for Edge Provider[1] v' as follows:

$$\min \sum_{j \in \mathcal{M}_i} \sum_{v \in \mathcal{N}} x_{j,v}^i \mathcal{C}(j, v) \tag{3a}$$

$$\text{s.t.} \sum_{j \in \mathcal{M}_i} x_{j,v}^i \rho_j^r \leq Crv \ \forall v \in \mathcal{N}, \forall r \in \Gamma \tag{3b}$$

$$\sum_{j \in \mathcal{M}_i} \sum_{k \in \mathcal{M}_i} x_{j,v}^i \sigma_{j,k} \mathbb{1}_{\{x_{k,v}^i \neq 1\}} \leq Nv \quad \forall v \in \mathcal{N} \tag{3c}$$

$$\sum_{v \in \mathcal{N}} x_{j,v} = 1 \forall j \in \mathcal{M}_i \tag{3d}$$

where constraint (3b) ensures that the computing budget of every cluster is not exceeded by deployed components, (3c) enforces the communication budget over networking demands between components that are deployed on different clusters, while (3d) ensures that all components are deployed.

Note that Problem (3) is a variant of the $|\Gamma|$-dimensional multi-Knapsack problem with *bin packing* [49], that is, all items must be assigned minimizing a cost function.

Lemma 1: Problem (3) is NP-Hard.

Proof: To demonstrate the complexity of Problem (3), we show that it can be reduced from the Partition Problem [50], which is known to be NP-hard. Given a set of positive integers $a_1, a_2, \ldots, a_n$, the Partition Problem consists of dividing them into two subsets such that the sum of the integers in each subset is equal. We create a simplified instance of Problem (3) as follows: consider only two identical clusters v' and v'', each featuring the same budget $C_{v'} = C_{v''} = \frac{1}{2} \sum_{j=1}^n a_j$ of a single resource type; assume the unitary price of such resource is 1, i.e., $p_{v'} = p_{v''} = 1$; consider an application with n components, where component j has a 1-dimensional computing demand $\rho_j = a_j$ and no demands in terms of networking. Therefore, the deployment cost of component j numerically coincides with its demand, i.e., $\mathcal{C}(j, v') = \mathcal{C}(j, v'') = a_j$. Note that, if there exists a partition of integers $a_1, a_2, \ldots, a_n$ into two equal-sum subsets $\{S_1, S_2\}$, then there exists a solution to Problem (3) (i.e., assign the components corresponding to S_1 to one cluster and those corresponding to S_2 to the other). This solution is also optimal since all the unitary prices are the same. Conversely, solving Problem (3) leads to a valid solution to the Partition Problem. Hence solving Problem (3) is at least as hard as solving the Partition Problem. ■

IV. PHARE ALGORITHM

This section describes a heuristic we designed to solve the NP-hard Problem (3a). We first provide some intuitions of what are the main challenges when scheduling components in distributed constrained infrastructure, and of the main concepts behind the algorithm logic. Then, we describe the algorithm and detail its steps.

[1] Since our algorithm operates in a decentralized fashion, we formulate the problem from the point of view of a certain edge provider v' and omit the under script $\cdot_{v'}$ for simplicity.

A. MAIN CHALLENGE

Decentralized allocation policies distribute decision-making among multiple agents, which improves scalability and resilience compared to centralized allocation. However, coordination and achieving global optimization can be challenging; the quality of the allocation will be discussed in the evaluation section, but it is important to note that coordination among agents plays a crucial role in the scalability of the solution. When an agent receives a request to deploy an application, it can independently perform the allocation process without exchanging information with other agents, which drastically reduces the need for synchronization. However, for informed decisions to be made, clusters must share their status with other members of the federation. A detailed description of the real-world implications of this will be discussed in Section V-G after the evaluation.

When deciding to schedule a particular component on a given cluster, a key role is played both by how big the component is (i.e., how many cluster resources it demands) and by how much it communicates with other components of the same application. A component that requires a lot of computational resources will be harder to schedule (it has less feasible matches) compared to small components, but this is also true for small components that feature intensive mutual communication (e.g., if the chosen host cluster has not enough bandwidth, the communication with any component placed outside will not occur properly).

Since edge infrastructures are highly scattered and constrained, we argue that it is particularly challenging to jointly satisfy the communication and computational requirements of all application components. Intuitively, the more components are scheduled on available clusters, the harder it becomes to schedule the remaining ones. Therefore, to quickly converge to a feasible placement, the algorithm should prioritize "harder" components, i.e., the ones featuring more stringent constraints (both in terms of computing and communication). With this intuition, we design our algorithm with the idea of guessing a convenient order for placing components, which would (*i*) minimize the chances of unfeasible deployments (quick convergence), and (*ii*) seek cost minimization.

B. ALGORITHM OVERVIEW

Our heuristic performs the steps in Algorithm 1.

When a request for deploying a new application $i \in \mathcal{I}$ is received, we first evaluate every component $j \in \mathcal{M}_i$ of application i and assign an *importance* metric z_j to each of them (Algorithm 1, line 3).[2] The higher the importance, the more the component is considered "hard to schedule", hence it will get a higher priority in the scheduling process. Details of how we estimate the importance of each component are provided in Section IV-C

[2]In practice, we compute separate values of importance z_j^r for each computing resource r, and then combine them back in Algorithm 1, line 10.

Algorithm 1 PHARE Scheduling for Application i

Require: $\mathcal{M}_i$, ρ_j^r $\forall j \in \mathcal{M}_i$, $\sigma_{j,k}$ $\forall j, k \in \mathcal{M}_i$

1: **for** $j \in \mathcal{M}_i$ **do**
2: **for** $r \in \Gamma$ **do**
3: Compute importance z_j^r of component j through (5)
4: **for** $v \in \mathcal{N}$ **do**
5: Estimate computing affinity $\Phi_{j,v}^r$ through (7)
6: Compute coefficient $a_{j,v}^r$ through (9)
7: Use a^r to estimate comm. affinity $\Psi_{j,v}^r$ through (8)
8: **end for**
9: **end for**
10: Combine per-resource importance into z_j through (6)
11: Take $\Phi_{j,v} \leftarrow \min_{r\in\Gamma}(\Phi_{j,v}^r)$, and $\Psi_{j,v} \leftarrow \min_{r\in\Gamma}(\Psi_{j,v}^r)$ (6)
12: **end for**
13: $J_S \leftarrow$ sort $\mathcal{M}_i$ by importance z_j descending
14: **for** $j \in J_S$ **do**
15: Schedule j to cluster $v^* \leftarrow \arg\min_{v\in\mathcal{N}} (C(j, v)/(\Phi_{j,v}\Psi_{j,v}))$
16: **end for**

After evaluating the importance of a component j, an affinity score is computed for each pair (j, v) of component j and feasible cluster $v \in \mathcal{N}$. The affinity provides an indication of how convenient it is to assign component j to cluster v, with respect to a trade-off between convergence speed and optimality of the final scheduling decision. In particular, two separate affinities $\Phi_{j,v} \in [0, 1]$ (*computing affinity* – Section IV-D) and $\Psi_{j,v} \in [0, 1]$ (*communication affinity* – Section IV-E) are estimated and combined (Algorithm 1, lines 5 - 7). We detail how we estimate computing and communication affinities in Sections IV-D and IV-E respectively.

We then compute costs $\mathcal{C}(j, v)$ for every component j and feasible cluster v, i.e., the marginal cost that would be required if j is scheduled on v. Such raw costs are adjusted using the affinity values computed at the previous step, thus obtaining the so-called *perceived cost* $\mathcal{C}(j, v)/(\Phi_{j,v}\Psi_{j,v})$: the less the affinity between component j and cluster v, the higher the perceived cost of scheduling j on v.

Finally, the algorithm iterates over components sorted by their importance z_j (descending), i.e., prioritizing those that feature stricter requirements, and assigns each component to the cluster v^* providing the less perceived cost (Algorithm 1, line 15), evaluated as

$$v^* = \arg\min_{v\in\mathcal{N}} \big(C(j, v)/\Phi_{j,v}\Psi_{j,v}\big). \tag{4}$$

In the remainder of this section, we complement the algorithm description by providing details for the missing pieces, namely, how we estimate components *importance* z_j, *computing* and *communication affinities* between components and clusters. Finally, we describe how we deal with multiple computing resources.

C. SORTING COMPONENTS BY IMPORTANCE

For each computing resource $r \in \Gamma$, we evaluate the importance of a component j mainly based on its demand ρ_j^r. This value is combined with the demands ρ_k^r of each "neighbor" component k, i.e., all those components

that feature some communication constraint with j. By prioritizing components with "heavy" neighbors we increase the probability that such neighbors are scheduled on the same clusters.

Before describing how we compute the importance z_j^r of a component j with respect to resource r, we provide the following definition of *communication factor*.

Definition 1 (Communication factor $\theta_{j,k}$): Given an application $i \in \mathcal{I}$ and two of its components $j, k \in \mathcal{M}_i$, we define $\theta_{j,k} = \sigma_{j,k} / \max_{j',k' \in \mathcal{M}_i}(\sigma_{j',k'})$ the communication factor between components j and k of application i.

The communication factor $\theta_{j,k}$ is an indicator of how intense is the communication demand between components j and k. We use this value to weight the contribution of each neighbor of j when estimating the importance z_j^r, as defined below.

Definition 2 (Importance z_j^r): Given component $j \in \mathcal{M}_i$ of an application $i \in \mathcal{I}$, we define the importance of component j with respect to resource r as

$$z_j^r = \rho_j^r + \sum_{k \in \mathcal{M}_i \setminus \{j\}} \theta_{j,k} \rho_k^r. \tag{5}$$

Algorithm 1 uses z_j^r to sort the application components so that those featuring more stringent deployment constraints are scheduled first. Specifically, given the multi-dimensionality of computing resources, the different values of resource importance are then combined as follows:

$$z_j = \sum_{r \in \Gamma} \frac{z_j^r}{\max_{r \in \Gamma}\left(\rho_j^r\right)}. \tag{6}$$

Intuitively, components that feature *(i)* high computing demands, *(ii)* neighbors with high computing demands, and *(iii)* high communication demands will be characterized by high importance values. Aside from determining the scheduling order, value z_j^r is also used to compute the affinity between component j and the available clusters, as described in the next section.

D. AFFINITY BETWEEN COMPONENTS AND CLUSTERS: COMPUTING

The first affinity factor we estimate only takes into account the computing resources of the target cluster, without considering its communication capabilities. To estimate the affinity between component j and cluster v, we first evaluate the quantity $y_j^r - \delta_v^r$, where δ_v^r is the residual computing resource r on cluster v, while y_j^r is the amount of overall computing resource of type r required by j and all its neighbors, i.e., $y_j^r = \rho_j^r + \sum_{k \in \mathcal{M}_i \setminus \{j\}} \rho_k^r \mathbb{1}_{\{\sigma_{j,k} > 0\}}$. We provide an intuitive definition of quantity $y_j^r - \delta_v^r$:

Definition 3 (Resource scarcity $y_j^r - \delta_v^r$): Let us assume component j is considered for deployment on cluster v. We define the difference between the amount of r-type resource required by j and all its neighbors and those still available on cluster v as resource scarcity $y_j^r - \delta_v^r$.

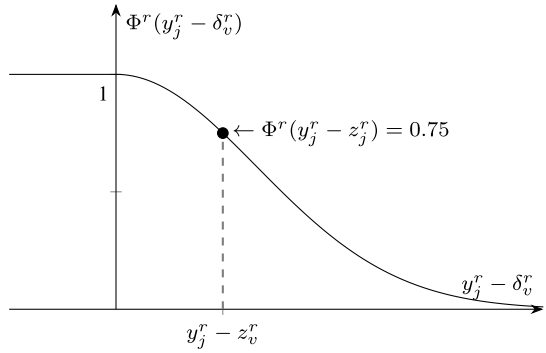


FIGURE 2. The computing affinity Φ^r varies with the resource scarcity $y_j^r - \delta_v^r$, i.e., the difference between the requirements of j and residual resources on cluster v. The critical value 0.75 is reached when v has enough resources to host a "significant portion" of j's neighborhood.

Note that when resource scarcity is less than zero, the cluster has enough type-r resources to accommodate j and its whole neighborhood.

The computing affinity is then evaluated based on the resource scarcity $y_j^r - \delta_v^r$ as follows:

$$\Phi_{j,v}^r = \begin{cases} e^{-c\frac{y_j^r - \delta_v^r}{y_j^r - z_j^r}} & \text{if } y_j^r - \delta_v^r > 0, \\ 1 & \text{if } y_j^r - \delta_v^r \leq 0. \end{cases} \tag{7}$$

where the coefficient c is used to adjust how fast the affinity decreases with respect to the lack of resources in the cluster.

To understand the rationale behind Equation (7) it is helpful to visualize the relationship between $\Phi_{j,v}^r$ and the quantity $y_j^r - \delta_v^r$ (Fig. 2). We provide an intuition below.

When $y_j^r - \delta_v^r \leq 0$, i.e., cluster v has enough resources to host j and all its neighbors, then the affinity is set to 1 (maximum affinity value). If the resources on v are not enough for hosting j and its whole neighborhood, the affinity starts to drop slowly, until the quantity $y_j^r - \delta_v^r$ reaches a critical value that leads to $\Phi_{j,v}^r = 0.75$; we set coefficient c so that this happens when $y_j^r - \delta_v^r = y_j^r - z_j^r$, i.e., when the residual resources on v are numerically equal to the importance z_j^r of component j.[3] The importance value here is used to estimate the portion of the neighborhood that is more significant for j: if the cluster has enough resources for hosting j and a significant portion of its neighborhood, then the affinity $\Phi_{j,v}^r$ will be higher than 0.75. Finally, after the critical value where $y_j^r - \delta_v^r = y_j^r - z_j^r$ is reached, the affinity $\Phi_{j,v}^r$ starts to drop quickly, with values eventually approaching 0.

E. AFFINITY BETWEEN COMPONENTS AND CLUSTERS: COMMUNICATION

To also take into account the networking capabilities of the target cluster, the computing affinity $\Phi_{j,v}^r$ is used in combination with a *communication affinity*. It is important to note that a task j will consume the communication capabilities (e.g., bandwidth) of the host cluster v only if its neighbors have been placed on some external clusters other

[3] This is achieved using $c = 0.287682$.

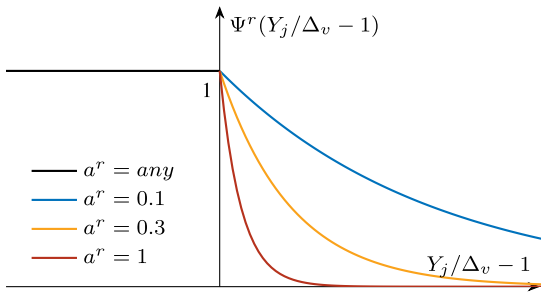


FIGURE 3. Bandwidth affinity for different values of the coefficient a^r.

than v, since otherwise, j would not need v's bandwidth to communicate with them. For this reason, when designing the communication affinity, we seek a mechanism that reduces the affinity of component j and cluster v the more it is difficult to accommodate the communication demands of j, but that has a lower impact if cluster v is large enough for potentially hosting a significant portion of j's neighborhood.

To calculate the communication affinity we first evaluate the quantity $Y_j/\Delta_v - 1$, where Δ_v is the residual communication capacity on cluster v, while Y is the overall communication demands for component j towards all its neighbors, i.e., $Y_j = \sum_{k \in \mathcal{M}_i \setminus \{j\}} \sigma_{j,k}$. Note that the quantity $Y_j/\Delta_v - 1$ is equal to zero when $Y_j = \Delta_v$.

The communication affinity is evaluated as follows

$$\Psi^r_{j,v} = \begin{cases} e^{-a^r(Y_j/\Delta_v - 1)} & \text{if } Y_j/\Delta_v - 1 > 0, \\ 1 & \text{if } Y_j/\Delta_v - 1 \leq 0, \end{cases} \tag{8}$$

where coefficient a^r is used to adjust the weight of the communication affinity so that it has a lower impact if the target cluster v has enough r-resources to host a significant portion of j's neighborhood: the higher coefficient a^r is, the more the communication affinity will affect the final solution (see below for details on how we compute coefficient a^r).[4]

The relationship between $\Psi^r_{j,v}$ and the quantity $Y_j/\Delta_v - 1$ is visualized in Figure 3. When cluster v has enough communication resources to accommodate all the communication demands of component j, the communication affinity is set to its maximum value 1. If the resources are not enough (i.e., the residual bandwidth is less compared to what j needs to communicate with the other components of the application), $\Psi^r_{j,v}$ drops exponentially with a decreasing factor that is based on the coefficient a^r: for higher values of a^r, the value of $\Psi^r_{j,v}$ drops more quickly.

Coefficient $a^r_{j,v}$: We compute the coefficient $a^r_{j,v}$ so that $\Psi^r_{j,v}$ decreases more slowly the more cluster v is likely to host some neighbors of j. The rationale is that if cluster v has enough computing resources to host a subset of j's neighborhood, then it is unfair to decrease the affinity between v and j based on v's communication capabilities (as

[4]Note that $a^r_{j,v}$ is different for each computing resource r; hence, we estimate multiple communication affinities $\Psi^r_{j,v}$, each associated with a certain computing resource r. Section IV-F describes how we deal with multiple computing resources.

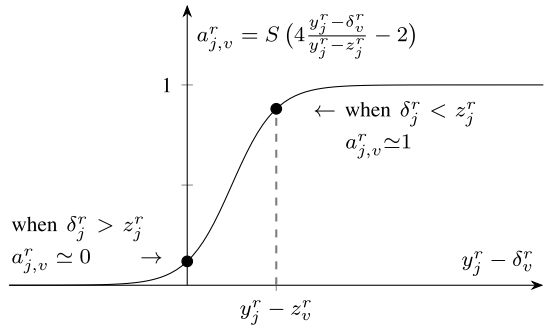


FIGURE 4. Coefficient a^r is determined based on the quantity $y^r_j - \delta^r_v$, i.e., its impact increases with the resource scarcity.

j will probably not need them). We empirically compute it as

$$a^r_{j,v} = S\left(4\frac{y^r_j - \delta^r_v}{y^r_j - z^r_j} - 2\right), \tag{9}$$

where S is the sigmoid function $S(x) = 1/(e^{-x} + 1)$. The relationship between coefficient $a^r_{j,v}$ and the quantity $y^r_j - \delta^r_v$ is visualized in Figure 4. If cluster v has enough r-resource compared to the demand of j and its neighborhood, then $a^r_{j,v} \simeq 0$, i.e., the communication affinity will have a negligible impact. If residual resources are not enough for hosting j and a significant portion of its neighborhood (estimated with z^r_j), then $a^r_{j,v} \simeq 1$ and the communication affinity will have maximum impact.

F. DEALING WITH MULTIPLE COMPUTING RESOURCES

Until now, we generalized the concept of computing resources since they experience similarities both in terms of provisioning and cost modeling. Nevertheless, realistic application deployments often feature two or even more conjoined computational constraints; we now describe how multiple resource constraints can be combined within our heuristic.

The values of computing and communication affinity defined respectively at 7 and 8 indicate the confidence in scheduling a component on a given cluster. Multiple computational constraints lead to multiple $\Phi^r_{j,v}$ and $\Psi^r_{j,v}$ per component j, i.e., one for each type of resource $r \in \Gamma$. We generalize the overall computing and communication affinities between component j and cluster v as

$$\Phi_{j,v} = \min_{r \in \Gamma}\left(\Phi^r_{j,v}\right), \text{ and } \Psi_{j,v} = \min_{r \in \Gamma}\left(\Psi^r_{j,v}\right), \tag{10}$$

extracting the minimum affinity value among the existing resources, hence considering the most conservative scenario.

V. EXPERIMENTAL RESULTS

In this section, we demonstrate the experimental validation of Phare. First, we aim to comprehend the impact of each mechanism composing Phare on job scheduling time and bandwidth usage. Second, we evaluate the scalability properties concerning infrastructure size (Section V-D) and the number of deployed applications (Section V-E). We

consider success rate, scheduling time, and the solution cost as key metrics for the subsequent evaluation. Ultimately, our assessment centers on the impact of Phare placement on network congestion within the infrastructure. This is achieved through a thorough analysis of the bandwidth consumption of scheduling solutions (Section V-F). We compare our results against Firmament, the state-of-the-art for microservice placement in cloud infrastructures.

A. IMPLEMENTATION OF THE SCHEDULING FRAMEWORK

We implemented a prototype version of Phare using the Golang language.[5] We designed the framework to be easily extensible to integrate and test additional scheduling algorithms in the same conditions.

The scheduling framework operates on a simulated environment in which both the infrastructures and the component-based applications are represented as data structures stored in memory. These can be either imported, to replicate specific scenarios, or be randomly generated for testing purposes. Particularly, the random generation is performed through configuration files that define boundaries for each type of resource both for application demands and for infrastructure availability.

The implementation of the scheduler interface for Phare relies on recursion to replicate the algorithm described in Section IV. Specifically, the recursive implementation extends the core algorithm, allowing it to probe multiple search paths, until a feasible solution is found, or a predefined timeout is triggered. Due to the design of Phare, the recursion tends to prioritize those initial placements that are more likely to lead to a feasible allocation, while jointly minimizing costs.

Using this framework, we also implement Firmament [22], a state-of-the-art scheduler and one of the few ones available in Kubernetes [51]. We aim to demonstrate that well-established Cloud solutions are far from behaving effectively when trivially adapted to the Edge scenario. Firmament exploits flow network representation of the scheduling problem to identify suitable placements by means of Flowlessly,[6] an efficient minimum-cost-maximum-flow decision problem solver. Our implementation of Firmament first translates the internal representation of both the infrastructure and the applications to the corresponding flow network (according to the specification provided in [22], [52], [53]), then calls the Flowlessly C++ library to solve the associated minimization problem.

Performance enhancement and additional features: Phare has been designed to seek early the most promising steps in the recursive process, but still the worst-case complexity can be estimated as $\mathcal{O}(N * J)$, where N is the number of clusters and J the number of components of the application to be scheduled. Such a worst-case scenario requires the simultaneous occurrence of numerous factors including huge application size and massive, almost-saturated, infrastructures. Although such a scenario is theoretically possible, providers usually prevent the saturation of their infrastructure for resilience reasons; still, the worst-case complexity can be reduced to $\mathcal{O}(M * J)$, with M being only some of the N feasible clusters, as empirical evaluations have shown that the scheduling solution is always found within the first $M = 10$ clusters, or not found at all. Consequently, we improve the sorting algorithm used to rank the most promising clusters accordingly: in particular, we use a modified version of Heap Sort that runs the Selection Sort only for the first M clusters. This reduces its complexity from $\mathcal{O}(N \log N)$ to $\mathcal{O}(N)$ as $M << N$.

Additional implemented features target the deployment on real systems: *(i)* define thresholds both for computational and network resource usage to prevent saturation, *(ii)* constraint the placement of a subset of the application components onto specific clusters to replicate given execution requirements, and *(iii)* define shared network links between clusters.

TABLE 1. Infrastructure setup.

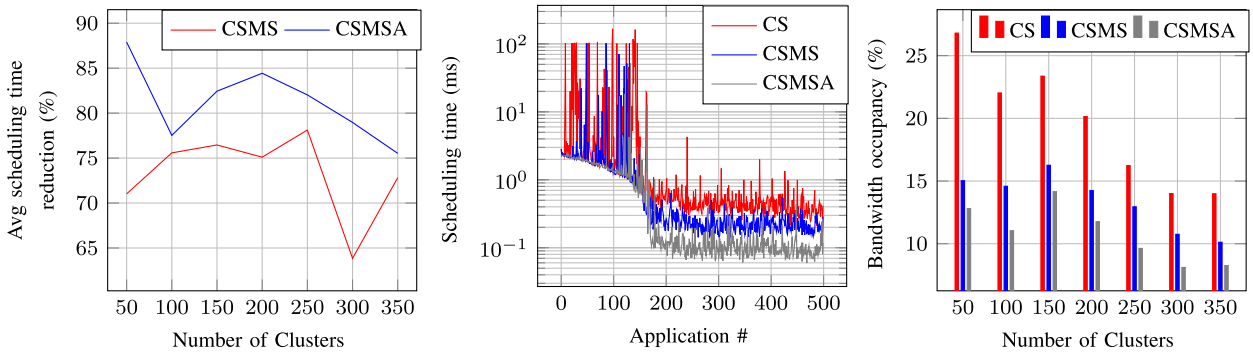
	Cluster type A	Cluster type B
Number of vCPU:	300 – 500 cores	4 – 32 cores
Cost of vCPU:	0.15 – 0.4 $\$/core$	0.15 – 0.4 $\$/core$
Available RAM:	256 – 1024 GB	8 – 64 GB
Cost of RAM:	0.01 – 0.06 $\$/GB$	0.01 – 0.06 $\$/GB$
Bandwidth:	1 – 10 Gb/s	0.1 – 2 Gb/s
Cost bandwidth:	0.05 – 0.2 $\$/Gb$	0.05 – 0.2 $\$/Gb$

B. EXPERIMENT SETUP

In our tests, we simulate random edge infrastructure topologies of different sizes, with the number of clusters ranging between 50 and 1000. Each cluster in the infrastructure features a predetermined random amount of CPU cores and available memory. These clusters are linked together with virtual connections, i.e., they logically form a full mesh topology. Each connection is characterized by the available network bandwidth for inter-cluster communication. Moreover, each resource features a given cost expressed in $\$/unit$, properly sized to match major Cloud Provider resource costs (TABLE 1 summarizes the main values concerning the infrastructure configuration).[7]

We use a sample 10-tier microservices application called Online Boutique[8] for our simulated workload. The workload reflects common patterns and challenges in distributed systems, thereby providing a credible and realistic benchmark for the use case applications in edge computing environments. To define our workload accurately, we first monitored the CPU, RAM, and bandwidth usage of the microservices and recorded the resource demands (our findings are detailed in TABLE 2). To account for the unpredictable randomness of the infrastructures and simulated

[5]The code is available at https://github.com/liqotech/scheduling.
[6]https://github.com/ICGog/Flowlessly
[7]https://cloud.google.com/compute/all-pricing
[8]https://github.com/GoogleCloudPlatform/microservices-demo



(a) Scheduling time reduction. (b) Scheduling time. (c) Bandwidth consumption.

FIGURE 5. **(a) Impact of heuristics used in Phare on the reduction of the scheduling time, with respect to simple Cost-based Scheduling (CS), namely Cost-based Scheduling with Microservice Sort (CSMS) and Cost-based Scheduling with Microservice Sort and Affinities (CSMSA). (b) Breakdown of the scheduling time for the 50-cluster infrastructure. (c) Average bandwidth occupancy of the links connecting the clusters.**

TABLE 2. **Workload setup.**

Application size:	10 microservices
Microservice CPU requests:	0.2 – 1 vCPU
Microservice RAM demand:	0.1 – 0.5 GB
Microservice connectivity ratio:	35%
Microservice bandwidth requirements:	150 – 1500 kB/s

workload, for each of our tests, we run multiple simulations (referred to as *simulation samples*) with varying random configurations. This was necessary to obtain statistically significant data.

C. EVALUATION OF THE HEURISTIC COMPONENTS

In this section, we assess the effectiveness of the intuitions behind the proposed algorithm in determining proper placement for microservice-based applications. In practice, we evaluate the benefits that each main building block pieces of Phare bring to the scheduling tasks. We deploy 500 applications (modeled based on TABLE 2) to random infrastructures, with the number of clusters ranging between 50 and 350 (each modeled based on TABLE 1, column B).

For this set of experiments, we use a Cost-based Scheduling (CS) algorithm as our baseline. This algorithm ranks clusters based on their unitary resource costs and places the i-th microservice on the cluster with the lowest cost. We assess the effectiveness of sorting microservice based on component importance z (Definition 2) comparing CS with an extended version named Cost-based Scheduling with Microservice Sort (CSMS) that implements the sorting mechanism described in Section IV-C. Additionally, we evaluate the impact of introducing our affinity mechanism (described in Section IV-D, Section IV-E by means of the Cost-based Scheduling with Microservice Sort and Affinities (CSMSA). Notice that CSMSA is equivalent to our final proposed heuristic Phare. Fig. 5(a) shows the scheduling time reduction of the proposed enhancements against the baseline. Specifically, sorting based on importance z enables the scheduling algorithm to first place the most demanding microservices (i.e., the ones with the most stringent requirements), moving to the less demanding ones afterward. Such an approach leads to a significant reduction in the average scheduling time (between 65% and 78%), also shortening the time required to identify unfeasible placements. The affinity mechanism further improves the scheduling time: according to resource availability, it effectively weights inter-component dependencies and leads to better mapping of application components onto infrastructure clusters. Indeed, CSMSA features a scheduling time reduction between 76% and 88% in our experiments. Additionally, Fig. 5(b) shows in which condition the proposed enhancements contribute the most to the reduction of the scheduling time. It details the measured scheduling time for each of the 500 applications in the 50-cluster infrastructure. It is important to note that the 50-cluster infrastructure cannot accommodate all the applications that have been submitted (only around 150 applications will be scheduled successfully, while the rest will not be scheduled due to insufficient resources). The plot shows that in situations where there is a shortage of resources, both CSMS and CSMSA introduce huge improvements on the baseline, visibly reducing the scheduling time by effectively identifying unfeasible placements quickly; moreover when there are enough resources to accommodate all the applications (i.e., less than 150 applications), the proposed scheduling mechanisms leads to faster scheduling times on average (spikes of 100 ms are less frequent when using CSMS, and even almost disappear with CSMSA). Finally, the proposed mechanisms improve not only in terms of scheduling time but also in the quality of the final placement. We evaluate this in terms of bandwidth occupancy of the links connecting the clusters: Fig. 5(C) shows that the full algorithm (CSMSA) consistently outperforms other configurations and never exceeds 15% of bandwidth usage.

D. SCALABILITY ON INFRASTRUCTURE SIZE

We now compare our algorithm against Firmament [22]. First, we evaluate the behavior of the two algorithms when scaling horizontally on the infrastructure size. Specifically,

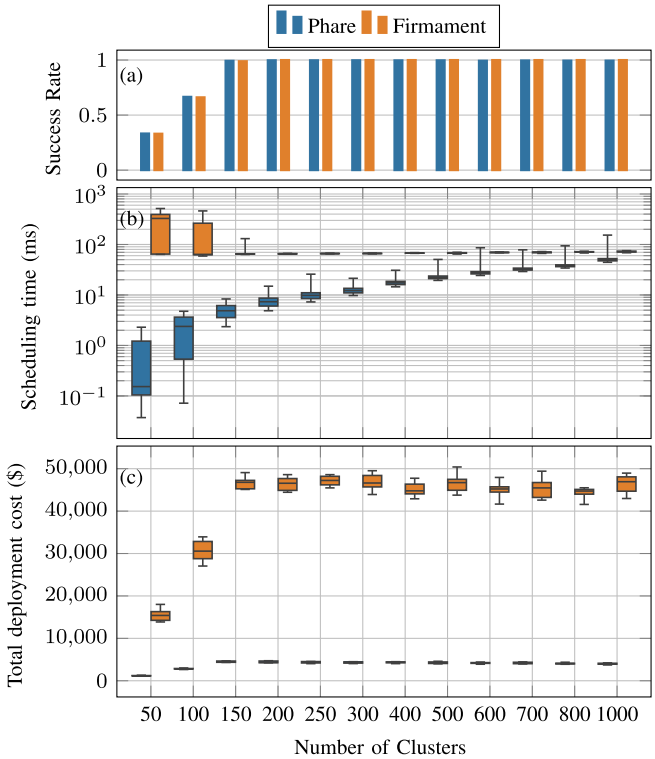


FIGURE 6. Respectively, scheduling success rate (Fig. 6a), scheduling time (Fig. 6b) and experienced costs for the computing resources reserved for the applications (Fig. 6c) for Phare and Firmament with infrastructures of variable sizes.

we run Phare and Firmament with infrastructures of multiple sizes, in order to study how the algorithms behave *(i)* when few (possibly saturated) clusters are available, i.e., limited scheduling options, and *(ii)* when many clusters are available (up to 1000 for our study case), with a large number of scheduling options to be evaluated.

Fig. 6a shows the fraction of successfully scheduled applications out of the complete set. An application is accounted as scheduled only if all its microservices have been successfully placed. Small infrastructures do not have enough resources to accommodate all the applications, hence infrastructures with less than 150 clusters experience a success rate lower than 1. On the other hand, larger infrastructures can host all the applications, hence a success rate is always equal to 1.

Results report a similar success rate for Phare and Firmament, i.e., both approaches consolidate almost the same amount of applications for each topology configuration. However, we experience a huge difference between the two algorithms in terms of both scheduling time and deployment cost. Fig. 6b represents the average time needed to fully schedule each of the 10K applications (the time needed to determine any unfeasible placement is accounted for as well). Phare largely outperforms Firmament, featuring less than 70 ms average scheduling time in large infrastructures, and even sub-millisecond scheduling time when operating on constrained topologies with few clusters. Conversely, Firmament experiences very high scheduling delays, especially on small infrastructures. This is because of its inability to quickly identify unfeasible application deployments, which is rare in cloud environments: by design, it accounts for scheduling only a subset of microservices per application at a time when the infrastructure is resource-constrained; this leads to multiple calls to the underlying solving algorithm.

Interestingly, the small scheduling time required by Phare w.r.t. Firmament does not have a negative impact on the experienced scheduling cost. Fig. 6c depicts the cumulative deployment cost of all the applications; not-scheduled applications contribute with a cost of 0 to the deployment cost. We can identify two trends throughout the test: *(i)* when considering small infrastructures with few federated clusters (less than 150), introducing additional clusters leads to a huge growth in the total deployment cost, as a considerable amount of new applications can be accommodated (as seen in Fig. 6a); *(ii)* infrastructure with more than 150 clusters have enough resources to host all the applications, hence the total deployment cost slightly decreases as the number of clusters — and the placement options — grow. In practice, the huge deployment cost gap between Phare and Firmament (up to 10×) is related to the cost of inter-microservice communication. Specifically, Firmament accounts for network bandwidth dependencies between application components but fails to correctly weight their cost effectively w.r.t. computing costs in a highly scattered topology. On the other hand, the network-aware placement of Phare can drastically reduce the bandwidth occupancy among different clusters and the associated networking costs. As a reference, the two algorithms experience similar costs for computing resources in this setting (result not shown). This result highlights the difficulties of adapting a Cloud-based algorithm to the Edge scenario, where network resources are far as uniform and optimally sized as they are in data centers.

E. SCALABILITY ON NUMBER OF APPLICATIONS

In the previous set of tests, we identified the 100-cluster infrastructure as one of the most challenging, due to the limited amount of available resources to fulfill the demands of all the applications. Analyzing closely such a scenario it is possible to understand the performance of the algorithms both when the infrastructure is almost saturated — and the possible placements are limited — but also when there are no more available resources in the infrastructure and the algorithm must quickly detect an unfeasible application placement.

Fig. 6b breaks down the scheduling success rate for the complete set of applications. As for the previous case, one application is accounted for as scheduled only if all its microservices have been successfully placed. The infrastructure does not have enough resources to accommodate all the applications, in fact, only the first 6K applications are always properly scheduled across all the simulation runs. Phare achieves a slightly higher success rate in the 7K-set of applications, scheduling a few more applications with respect

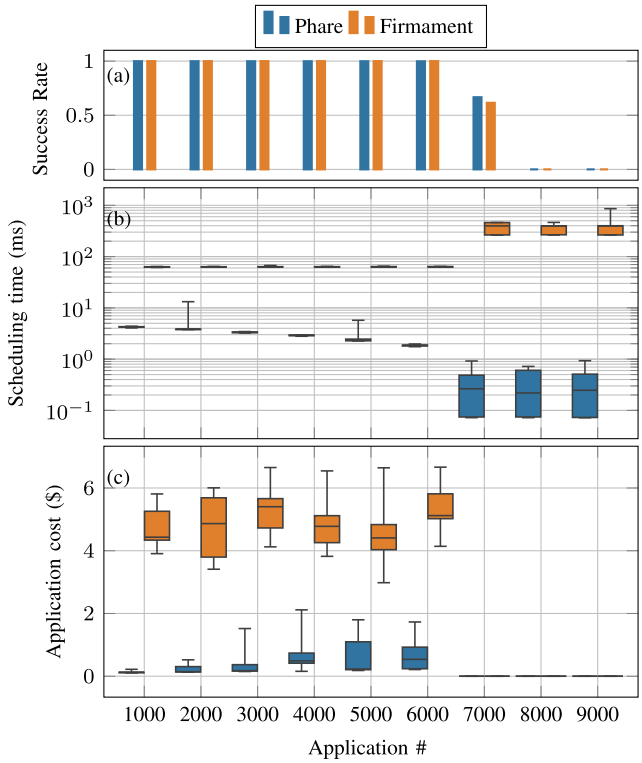


FIGURE 7. Scheduling success rate (Fig. 6b), scheduling time (Fig. 7b) and solution costs (Fig. 7c) for Phare and Firmament with infrastructures of 100 clusters.

to Firmament thanks to more accurate management of the available resources.

As for the previous test, the two algorithms heavily differ in measured scheduling time: Fig. 7b reports the observed scheduling time distribution for the i_{th} application. Phare outperforms Firmament being able not only to converge to a suitable scheduling solution in almost-saturated infrastructures but also to quickly detect unfeasible solutions when the amount of available resources is not enough to host the remaining applications. Conversely, Firmament experiences some performance drop with saturated infrastructures because of the same design issue mentioned in the previous section.

Finally, the two algorithms experience again similar costs for what concerns computing resources (not shown), but provide a huge gap when including the cost of networking resources (Fig. 7c). Indeed, Firmament behaves poorly when network resources are not uniform and networking costs become relevant, which is definitely the case in multi-cloud environments or, in general, for highly distributed infrastructures. In this case, we observe that Phare reduces deployment costs by more than 5 times overall compared to Firmament.

F. BANDWIDTH CONSUMPTION

We now evaluate the benefits of the communication-aware placement of Phare, thus breaking down the bandwidth consumption of the links between clusters.

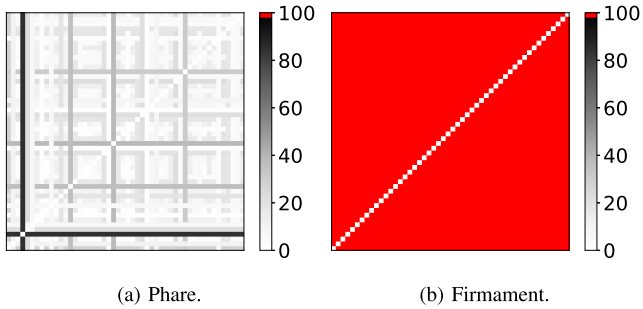


FIGURE 8. Inter-cluster congestion matrix: each cell depicts the percentage of network bandwidth usage between two clusters

For this specific set of tests, we use a 50-cluster infrastructure (dimensioned according to TABLE 1, column A). Fig. 8 depicts the measured network pressure on the link between clusters; specifically, they represent the congestion matrix $N * N$, where N is the number of clusters of the federation and each cell (i, j) represents the percentage of network bandwidth usage between cluster i and j. Moreover, each cell value is represented on a gray-scale color code, except the ones in red that exceed 100% network usage (i.e., the target placement requires more bandwidth than the available amount). In these extreme settings, Firmament even fails to find placements that are feasible from a networking perspective. On the other hand, Phare is able to intelligently distribute the applications across the federation, properly aggregating within the same clusters those microservices that feature mutual dependencies.

G. REVISIONREAL-WORLD IMPLEMENTATION CONSIDERATIONS

While our evaluation primarily focuses on the algorithm's performance within simulated environments, we acknowledge the importance of applicability and challenges in real-world settings. Besides scalability, there exist several critical areas to consider, which we briefly analyze below.

Integration with existing systems: Seamless integration with existing infrastructure and cloud/edge computing platforms is crucial for the adoption of any new technology. To this end, we anticipate the potential for integrating Phare in Kubernetes by replacing the default scheduling algorithm with our solution.

Forming a federation: In a federation, computing resources are shared between clusters, allowing each entity to access additional resources if needed. This resource sharing can be achieved through the multi-cloud functionalities provided by Liqo,[9] which is already being used as an enabling technology in the European project FLUIDOS[10] to create distributed continuum infrastructures.

Practical performance metrics: Phare requires updated information on the clusters participating in the federation, including CPU, memory, and bandwidth usage. A real-world

[9]https://liqo.io/
[10]https://www.fluidos.eu/

implementation should thus carefully balance the resolution of such data and the additional overhead required to process it. Specifically, whereas a peer-to-peer interaction to retrieve cluster usage metrics might be suitable for small federations, it does not scale with the number of clusters. Therefore, a central metrics aggregation point might be a suitable solution for a real implementation.

Privacy and security: Privacy and security are vital for ensuring that workloads are executed correctly in a distributed infrastructure. There are two major issues related to job allocation that must be addressed: First, the device hosting the container execution must provide a secure environment. This can be achieved by using the functionalities offered by the container runtime, Kubernetes, and Liqo. Second, the allocation process must take into account customer constraints. It is crucial to ensure that devices are authenticated through trusted parties to prevent allocation in insecure sites.

By addressing these considerations, we aim to bridge the gap between theoretical research and practical application, ensuring that our algorithm not only excels in simulated tests but also meets the demands of real-world deployment.

VI. CONCLUSION

Compared to the Cloud scenario, scattered and constrained clusters in Edge infrastructure pose non-trivial challenges for computing and bandwidth resource scheduling, which can be hardly addressed by adapting traditional data center techniques.

In this paper, we proposed a new approach to multi-cloud application scheduling, called Phare, that takes into account the communication and computing requirements of the entire application graph and leverages the capabilities of the underlying infrastructure to optimize resource usage. Phare is based on a heuristic to speed up the placement of the application while minimizing the overall application deployment cost. We compared Phare against Firmament, a state-of-the-art scheduler largely employed in cloud infrastructures, performing a series of tests over infrastructures of various sizes. Our results show that, despite featuring a similar scheduling success rate, Phare largely outperforms Firmament in terms of scheduling time, being able to quickly identify suitable placements even for large infrastructures. Modeling real major provider resource costs, we show that networking costs can be a major factor in highly distributed infrastructures, and assess Phare benefits over Firmament with up to 10× deployment cost reduction scheduling 10K applications on 1K clusters. Furthermore, our tests on network congestion show that Phare can effectively reduce the bandwidth usage between clusters, resulting in more efficient resource usage and better overall performance.

Overall, our results demonstrate the effectiveness of Phare in optimizing microservice application scheduling on multi-cloud highly-distributed infrastructures and suggest that it can be a valuable tool for organizations that rely on scattered edge resources to run their applications.

ACKNOWLEDGMENT

The authors warmly thank Gabriele Filaferro and Luca Nicosia for their precious help in implementing and validating the proposed algorithm.

REFERENCES

[1] "Kubernetes website." Kubernetes. 2021. [Online]. Available: https://kubernetes.io/

[2] "Apache Hadoop website." 2021. [Online]. Available: https://hadoop.apache.org/

[3] V. K. Vavilapalli et al., "Apache Hadoop YARN: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput.*, 2013, pp. 1–16.

[4] V. Dukic et al., "Beyond the mega-data center: Networking multi-data center regions," in *Proc. Annu. Conf. ACM Spec. Interest Group Data Commun. Appl., Technol., Archit., Protoc. Comput. Commun.*, 2020, pp. 765–781.

[5] K. Church, A. G. Greenberg, and J. R. Hamilton, "On delivering embarrassingly distributed cloud services," in *Proc. HotNets*, 2008, pp. 55–60.

[6] M. Iorio, F. Risso, A. Palesandro, L. Camiciotti, and A. Manzalini, "Computing without borders: The way towards liquid computing," *IEEE Trans. Cloud Comput.*, vol. 11, no. 3, pp. 2820–2838, Jul.–Sep. 2023.

[7] I. Narayanan, A. Kansal, A. Sivasubramaniam, B. Urgaonkar, and S. Govindan, "Towards a leaner geo-distributed cloud infrastructure," in *Proc. 6th USENIX Workshop Hot Topics Cloud Comput.*, 2014, p. 3.

[8] F. Lucrezia, G. Marchetto, F. Risso, and V. Vercellone, "Introducing network-aware scheduling capabilities in OpenStack," in *Proc. Proc. 1st IEEE Conf. Netw. Softw. (NetSoft)*, 2015, pp. 1–5.

[9] S. Galantino, F. Risso, A. Cazzaniga, F. Garrone, R. Terruggia, and R. Lazzari, "An edge-based architecture for Phasor measurements in smart grids," in *Proc. AEIT Int. Annu. Conf. (AEIT)*, 2022, pp. 1–6.

[10] M. A. Tamiru, G. Pierre, J. Tordsson, and E. Elmroth, "Instability in geo-distributed kubernetes federation: Causes and mitigation," in *Proc. 28th Int. Symp. Model., Anal., Simul. Comput. Telecommun. Syst. (MASCOTS)*, 2020, pp. 1–8.

[11] L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, and M. Kihl, "Impact of etcd deployment on kubernetes, Istio, and application performance," *Softw., Pract. Exp.*, vol. 50, no. 10, pp. 1986–2007, 2020.

[12] L. Osmani, T. Kauppinen, M. Komu, and S. Tarkoma, "Multi-cloud connectivity for Kubernetes in 5G networks," *IEEE Commun. Mag.*, vol. 59, no. 10, pp. 42–47, Oct. 2021.

[13] "Apache Flink website." 2021. [Online]. Available: https://flink.apache.org/

[14] "Apache storm website." 2021. [Online]. Available: https://storm.apache.org/

[15] "Apache Kafka website." 2021. [Online]. Available: https://kafka.apache.org/

[16] "Apache Solr website." 2021. [Online]. Available: https://solr.apache.org/

[17] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[18] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 613–627.

[19] X. Meng et al., "MLlib: Machine learning in Apache spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, 2016.

[20] P. Garefalakis, K. Karanasos, P. Pietzuch, A. Suresh, and S. Rao, "MEDEA: Scheduling of long running applications in shared production clusters," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–13.

[21] S. Li, L. Wang, W. Wang, Y. Yu, and B. Li, "George: Learning to place long-lived containers in large clusters with operation constraints," in *Proc. ACM Symp. Cloud Comput.*, 2021, pp. 258–272.

[22] I. Gog, M. Schwarzkopf, A. Gleave, R. N. Watson, and S. Hand, "Firmament: Fast, centralized cluster scheduling at scale," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 99–115.

[23] P. Delgado, D. Didona, F. Dinu, and W. Zwaenepoel, "Job-aware scheduling in eagle: Divide and stick to your probes," in *Proc. 7th ACM Symp. Cloud Comput.*, 2016, pp. 497–509.
[24] P. Delgado, D. Didona, F. Dinu, and W. Zwaenepoel, "Kairos: Preemptive data center scheduling without runtime estimates," in *Proc. ACM Symp. Cloud Comput.*, 2018, pp. 135–148.
[25] F. Giroire, N. Huin, A. Tomassilli, and S. Pérennes, "When network matters: Data center scheduling with network tasks," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 2278–2286.
[26] P. Thinakaran, J. R. Gunasekaran, B. Sharma, M. T. Kandemir, and C. R. Das, "Phoenix: A constraint-aware scheduler for heterogeneous datacenters," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2017, pp. 977–987.
[27] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 2287–2295.
[28] J. Rasley, K. Karanasos, S. Kandula, R. Fonseca, M. Vojnovic, and S. Rao, "Efficient queue management for cluster scheduling," in *Proc. 11th Eur. Conf. Comput. Syst.*, 2016, pp. 1–15.
[29] A. Chung, J. W. Park, and G. R. Ganger, "Stratus: Cost-aware container scheduling in the public cloud," in *Proc. ACM Symp. Cloud Comput.*, 2018, pp. 121–134.
[30] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in *Proc. ACM SIGOPS 22nd Symp. Oper. Syst. Princ.*, 2009, pp. 261–276.
[31] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with borg," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, pp. 1–17.
[32] L. A. Rocha and F. L. Verdi, "A network-aware optimization for VM placement," in *Proc. IEEE 29th Int. Conf. Adv. Inf. Netw. Appl.*, 2015, pp. 619–625.
[33] M. A. Abdelaal, G. A. Ebrahim, and W. R. Anis, "Network-aware resource management strategy in cloud computing environments," in *Proc. 11th Int. Conf. Comput. Eng. Syst. (ICCES)*, 2016, pp. 26–31.
[34] F. Larumbe and B. Sansò, "Elastic, on-line and network aware virtual machine placement within a data center," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manag. (IM)*, 2017, pp. 28–36.
[35] L. Wang, X. Deng, J. Gui, X. Chen, and S. Wan, "Microservice-oriented service placement for mobile edge computing in sustainable Internet of Vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 9, pp. 10012–10026, Sep. 2023.
[36] Z. Lin, S. Bi, and Y.-J. A. Zhang, "Optimizing AI service placement and resource allocation in mobile edge intelligence systems," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7257–7271, Nov. 2021.
[37] L. Chen, C. Shen, P. Zhou, and J. Xu, "Collaborative service placement for edge computing in dense small cell networks," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 377–390, Feb. 2021.
[38] X. Li, Z. Lian, X. Qin, and W. Jie, "Topology-aware resource allocation for IoT services in clouds," *IEEE Access*, vol. 6, pp. 77880–77889, 2018.
[39] B. Ryu, A. An, Z. Rashidi, J. Liu, and Y. Hu, "Towards topology aware pre-emptive job scheduling with deep reinforcement learning," in *Proc. 30th Annu. Int. Conf. Comput. Sci. Softw. Eng.*, 2020, pp. 83–92.
[40] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana, "E3: Energy-efficient microservices on SmartNIC-accelerated servers," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 363–378.
[41] F. Rossi, V. Cardellini, F. L. Presti, and M. Nardelli, "Geo-distributed efficient deployment of containers with Kubernetes," *Comput. Commun.*, vol. 159, pp. 161–174, Jun. 2020.
[42] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards delay-aware container-based service function chaining in fog computing," in *Proc. IEEE/IFIP Netw. Oper. Manag. Symp.*, 2020, pp. 1–9.
[43] B. Németh et al., "Fast and efficient network service embedding method with adaptive offloading to the edge," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2018, pp. 178–183.
[44] V. Sundararaj, "Optimal task assignment in mobile cloud computing by queue based ant-bee algorithm," *Wireless Pers. Commun.*, vol. 104, no. 1, pp. 173–197, 2019.
[45] G. Castellano, F. Esposito, and F. Risso, "A distributed orchestration algorithm for edge computing resources with guarantees," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 2548–2556.
[46] O. Ascigil, T. K. Phan, A. G. Tasiopoulos, V. Sourlas, I. Psaras, and G. Pavlou, "On uncoordinated service placement in edge-clouds," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, 2017, pp. 41–48.
[47] I. Alghamdi, C. Anagnostopoulos, and D. P. Pezaros, "Delay-tolerant sequential decision making for task offloading in mobile edge computing environments," *Information*, vol. 10, no. 10, p. 312, 2019.
[48] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Energy-aware application placement in mobile edge computing: A stochastic optimization approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 909–922, Apr. 2020.
[49] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Hoboken, NJ, USA: Wiley, 1990.
[50] R. E. Korf, "A complete anytime algorithm for number partitioning," *Artif. Intell.*, vol. 106, no. 2, pp. 181–203, 1998.
[51] D. Vij and S. Shrivastava. "Poseidon-firmament scheduler—Flow network graph based scheduler." Accessed: Aug. 23, 2021. [Online]. Available: https://github.com/kubernetes-sigs/poseidon.
[52] I. C. Gog, "Flexible and efficient computation in large data centres," Ph.D. dissertation, Comput. Lab., Univ. Cambridge, Cambridge, U.K., 2018.
[53] M. Schwarzkopf, "Operating system support for warehouse-scale computing," Ph.D. dissertation, Comput. Lab., Univ. Cambridge, Cambridge, U.K., 2018.

GABRIELE CASTELLANO received the M.Sc. degree in computer engineering and the Ph.D. degree from the Politecnico di Torino, Italy, in 2016, and 2020 respectively. He is currently a Postdoctoral Researcher with Nokia Bell Labs and Inria. His research interests include resource orchestration, distributed algorithms, and artificial intelligence.

STEFANO GALANTINO received the M.Sc. degree in computer engineering from the Politecnico di Torino, Italy, where he is currently pursuing the Ph.D. degree. His research interests include cloud-to-edge continuum, sustainable, and energy-efficient computing.

FULVIO RISSO received the Ph.D. degree in computer engineering from the Politecnico di Torino, Italy, where he is an Associate Professor. His research interests include high-speed and flexible network processing, edge/fog computing, and network functions virtualization.

ANTONIO MANZALINI received the Ph.D. degree in computer science from Sorbonne University. He is a Senior Project Manager with TIM, Turin, Italy. His research interests include edge computing and quantum communications.

Open Access funding provided by 'Politecnico di Torino' within the CRUI CARE Agreement